Integrating Process Learning and Process Evolution – A Semantics Based Approach

Stefanie Rinderle[1], Barbara Weber[2], Manfred Reichert[3], and Werner Wild[4]

[1] Dept. Databases and Information Systems, University of Ulm, Germany
rinderle@informatik.uni--ulm.de
[2] Quality Engineering Research Group, University of Innsbruck
Barbara.Weber@uibk.ac.at
[3] Information Systems Group, University of Twente, The Netherlands
m.u.reichert@cs.utwente.nl
[4] Evolution Consulting, Innsbruck, Austria
werner.wild@evolution.at

Abstract. Companies are developing a growing interest in aligning their information systems in a process-oriented way. However, current process-aware information systems (PAIS) fail to meet process flexibility requirements, which reduces the applicability of such systems. To overcome this limitation PAIS should capture the whole process life cycle and all kinds of changes in an integrated way. In this paper we present such a holistic approach providing full process life cycle support by combining the ADEPT framework for dynamic process changes with the concepts and methods provided by case-based reasoning (CBR) technology. This allows expressing the semantics of process changes, their memorization and their reuse to perform similar changes in the future. If the same or similar process instance changes occur frequently, potential process type changes are suggested to the process engineer. The process engineer can then perform a schema evolution and migrate running instances to the new schema version by using the ADEPT framework. Finally, the case–base related to the old schema version is migrated as well.

1 Introduction

Adaptive process management technology offers a promising approach for realizing highly flexible, process–oriented information systems [1,2,3,4]. In particular, it enables dynamic process changes during runtime to handle exceptional situations and changing needs. Basically, such process changes can be made at two levels – the *process type* and *process instance* level [5].

In our experience an adaptive process management system (PMS) must support both kinds of changes in an integrated way [6]. In the ADEPT project we have elaborated a conceptual framework which enables changes at the process instance and at the process type level. For the latter we support the subsequent migration of both *unbiased* and *biased* process instances to the changed process type schema. This is especially important for long–running process instances [7].

W.M.P. van der Aalst et al. (Eds.): BPM 2005, LNCS 3649, pp. 252–267, 2005.

We denote process instances as unbiased if they are running according to the original process schema they were derived from [8], whereas process instances are denoted as biased when they have been individually modified by an ad–hoc change [6].

So far, our work on adaptive processes (e.g., [7,8,9]) has not incorporated application semantics, i.e., it has not considered the reasons for and the context of a change. Instead, very similar to database technology, all checks and procedures necessary to perform a dynamic change have been applied solely at the syntactical level, which, nevertheless, is an important prerequisite for any adaptive process management technology. To provide more intelligent support to its users and to reuse knowledge about previously applied process changes *semantical* aspects must be considered as well. This paper shows how adding semantics contributes to the seamless integration of process changes into the process life cycle (cf. Fig. 1).

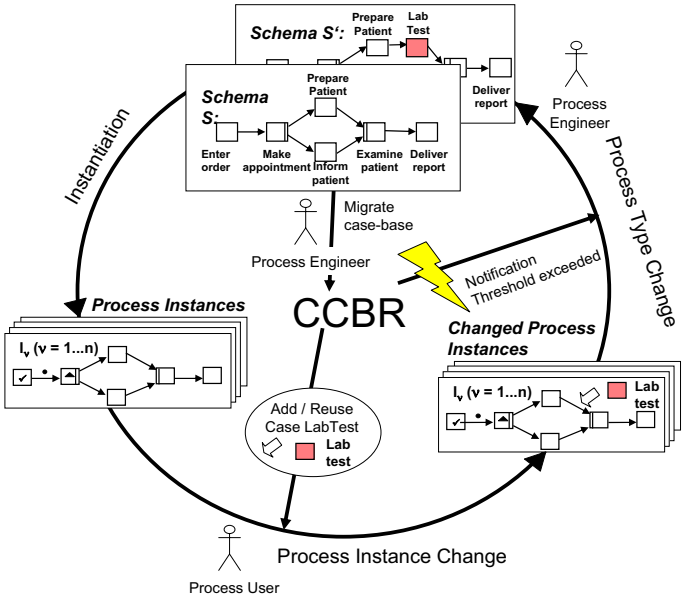

Fig. 1. Process Life Cycle

In this paper we combine the ADEPT framework for process changes with the concepts and methods provided by CBRFlow [10], a process change approach using case-based reasoning (CBR) technology. This allows us to express the semantics of process changes and to provide users with information about the context of and the reasons for process changes, ensuring the traceability of instance changes. The latter is a crucial requirement in many domains (e.g., hospitals have strict guidelines regarding the documentation of deviations from the predefined treatment process). Respective change information is stored as cases

in a process schema specific *case–base*. This information can be used to support process actors in reusing information about similar ad-hoc changes which have been applied to previously executed process instances. Change definition requires significant user experience, especially when further adaptations become necessary (e.g., when deleting a particular activity, data-dependent activities may have to be deleted as well [9]). Therefore, reuse of existing knowledge about previous ad-hoc changes is highly desirable.

Furthermore, case–bases are continuously monitored in order to automatically derive suggestions for process type changes from previously applied process instance changes. In practice, necessary type changes are frequently indicated by process instances whose execution deviates from the original process type schema over and over again. In such a situation a process type change is desirable to move the respective optimizations up to the process type level.

In the ADEPT framework a process type change is performed by deriving a new version of the process type schema and – if possible and desired – by automatically migrating the already running process instances to this new schema version [7,8]. This may include the migration of both unbiased and biased process instances. Interestingly, not only processes but also case-bases evolve over time. When a case–base is "migrated" to a new process schema version, it should only keep information which is still relevant for instances of the new process schema (and for changes to them). In our approach a process schema evolution is always accompanied by the evolution of the related case-base. This poses several challenges which will be discussed later in this paper.

In Section 2 we provide background information. Section 3 presents fundamentals of CBR and their application to process instance changes. In Section 4 we show how to learn from instance changes by using CBR techniques and we provide an overview of our process evolution approach. The evolution of case–bases is described in Section 5. We discuss related work in Section 6 and close with a summary and an outlook in Section 7.

2 Background Information

In this section we give basic definitions for process type schemes and process instances as needed for our further considerations. To improve readability we restrict the discussion to Activity Nets [11]; however, our approach is applicable to more complex process meta models as well (e.g., WSM Nets [7]).

For each business process to be supported a process type T is defined. It is represented by a *process type schema* which may exist in different versions.

Definition 1 (Process Type Schema). *A tuple S with S = (N, D, CtrlEdges, DataEdges, EC) is called a process schema, if the following holds:*

- *N is a set of activities and D a set of data elements*
- *CtrlEdges* $\subset N \times N$ *is a precedence relation*
  (notation: $n_{src} \rightarrow n_{dst} \equiv (n_{src}, n_{dst}) \in$ *CtrlEdges)*

- *DataEdges* $\subseteq N \times D \times \{$*read, write*$\}$ *is a set of read/write data links between activities and data elements*
- *EC: CtrlEdges* $\mapsto$ *Conds(D) where Conds(D) denotes the set of all valid transition conditions on data elements from D.*

For a process type schema several correctness constraints exist, e.g., (N, CtrlEdges) must be an acyclic graph to ensure the absence of deadlocks.

At runtime new process instances are created from a process schema S and are then executed. Each instance I is associated with an instance-specific schema $S_I := S + \Delta_I$[1]. S = S(T,V) denotes the original process schema S from which instance I was derived, whereby T is the process type of I and V the version of the process type schema. $\Delta_I = \{op_1, ..., op_n\}$ comprises the change operations op_i (i = 1, ..., n) applied to I so far (cf. Fig. 4). In this context a change operation $op_i = (opType, s, paramList)$ (i = 1, ..., n) is specified by an operation type (e.g., insertion of an activity), a subject (e.g., the newly inserted activity), and a list of parameters (e.g., position of the newly inserted activity). Selected change operations are shown in Table 1.

The execution state of I is captured by marking function $\mathrm{M}^{S_I} = (\mathrm{NS}^{S_I}, \mathrm{ES}^{S_I})$. It assigns to each activity n its current status $NS(n)$ and to each control edge e its marking $ES(e)$. Markings are determined according to well defined rules, markings of already passed regions and skipped branches are preserved.

Definition 2 (Process Instance). *A process instance I is defined by a tuple (T, V,* Δ_I*,* M^{S_I}*,*Val^{S_I}*) where*

- *T denotes the process type of I and V the version of the process schema S := S(T,V) = (N, D, CtrlEdges, ...) instance I was derived from. We call S the original schema of I.*
- Δ_I *comprises the instance-specific changes* $op_1, \ldots, op_m$ *that have been applied to I so far*[2]*. We call* Δ_I *the bias of I. Schema* $S_I := S + \Delta_I$ *(with* $S_I = (N_I, D_I, \ldots)$*) which results from the application of* Δ_I *to S, is called the instance–specific schema of I.*
- M^{S_I} *=* (NS^{S_I}, ES^{S_I}) *describes node and edge markings of I:*
  NS^{S_I}*:* $N_I \mapsto$ {`NotActivated, Activated, Running, Completed, Skipped`}
  ES^{S_I}*:* $(CtrlEdges_I) \mapsto$ {`NotSignaled, TrueSignaled, FalseSignaled`}
- Val^{S_I} *is a function on* D_I*. It reflects for each data element* $d \in D_I$ *either its current value or the value* `UNDEFINED` *(if d has not been written yet).*

Table 1 presents a selection of *high-level change operations* on process schemes which can be used to create or modify schemes at the type as well as at the instance level. These change operations also include formal pre- and

[1] For unbiased instances $\Delta_I(S) = \emptyset$ and consequently $S_I = S$ holds.

[2] Thereby an operation $op_j := (opType_j, s_j, paramList_j)$ (j = 1, ..., m) consists of an operation type opType_j, the subject s_j of the change, and a parameter list $\mathrm{paramList}_j$ (cf. Tab. 1).

post-conditions. They automatically perform the necessary schema transformations while ensuring schema correctness. An example for such a change operation is the insertion of an activity and its embedding into the process context.

Table 1. *A Selection of High-Level Change Operations on Process Schemes*

Change Operation *op* **Applied to S**	opType	subject	paramList	**Effects on S**
		Additive Change Operations		
sInsert(S, X, A, B)	Insert	X	S, A, B	inserts activity X between two directly succeeding activities A and B
cInsert(S, X, A, B, sc)	Insert	X	S, A, B, sc	inserts activity X between two directly succeeding activities A and B as a new branch with selection code sc; edge (A, B) gets selection code "default"
		Subtractive Change Operations		
delAct(S, X)	Delete	X	S	deletes activity X from schema S
		Order-Changing Operations		
sMove(S, X, A, B)	Move	X	S, A, B	moves activity X from its current position to position between directly succeeding activities A and B

3 Providing Change Semantics Through CCBR

In this section we describe how CBR can be used to capture the semantics of process changes, how these changes can be memorized, and how they can be retrieved and reused when similar changes become necessary in the future.

3.1 Introduction to Case-Based Reasoning

Case-based reasoning (CBR) is a contemporary approach to problem solving and learning [12]. New problems are dealt with by drawing on past experiences – described in cases and stored in case–bases – and by adapting their solutions to the new problem situation. Reasoning by using past experiences is a powerful and frequently applied way to solve problems by humans [13]. A physician, for example, remembers previous cases to determine the disease of a new patient. A banker working on a difficult loan decision uses her experiences about previous cases in order to decide whether to grant a loan or not.

A case is a contextualized piece of knowledge representing an experience [12], which typically consists of a problem description and the corresponding solution. As opposed to most other approaches in Artificial Intelligence, CBR uses specific knowledge of past experiences to solve new problems. CBR also contributes to incremental and sustained learning: every time a new problem is solved, information about it and its solution is retained and therefore immediately made available for solving future problems [13].

Conversational CBR (CCBR) is an extension of the CBR paradigm, which actively involves users in the inference process [14]. A CCBR system can be characterized as an interactive system that, via a mixed-initiative dialogue, guides users through a question-answering sequence in a case retrieval context (cf.

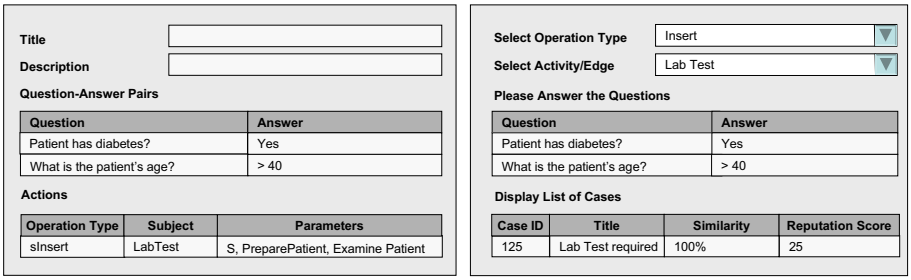

Fig. 2. CCBR User Dialogs - Adding a New Case and Retrieving Similar Cases

Fig 2). Unlike traditional CBR, CCBR neither requires the user to provide a complete a priori problem specification for case retrieval nor requires him to provide knowledge about the relevance of each feature for problem solving. Instead, the system assists the user in finding relevant cases by presenting a set of questions to assess the given situation. It guides users who can supply already known information on their initiative. Therefore, CCBR is especially suitable for handling exceptional or unanticipated situations that cannot be dealt with in a fully automated way.

3.2 Conversational Case-Based Reasoning and Adaptive Workflows

In our approach CCBR is used to provide semantical information about changes to process instances. This information can be reused when either similar ad-hoc modifications become necessary or to trigger process type changes.

Case Representation. In our context, a case c represents a concrete ad-hoc modification to one or more process instances providing the context of and the reasons for the deviation (cf. Fig. 2). It consists of a textual problem description *pd* which briefly describes the exceptional or unanticipated situation which made the ad-hoc modification necessary. The reasons for the change are described as question-answer pairs $\{q_1an_1, \ldots, q_nan_n\}$. Each question-answer pair denotes a condition under which the modification has become necessary; they are used to retrieve similar cases when a problem arises in the future. The solution part *sol* (i.e., the list of actions) contains the change operations (and related context information) to be executed to deal with the exceptional or unanticipated situation. Finally, the reputation score *rScore* of a case indicates how successfully it has been reused in the past, i.e., the trust users can have into the semantical correctness of this case. The reputation score is calculated as the sum of the feedback scores (see below).

Definition 3 (Case). *A case c is a tuple (pd,* $\{q_1an_1, \ldots, q_nan_n\}$*, sol, rScore) where*

- *pd is a textual problem description*
- $\{q_1an_1, \ldots, q_nan_n\}$ *denotes a set of question-answer pairs*

- *sol* = { op_j | $op_j = (opType_j, s_j, paramList_j)$, *j = 1, ..., k*} *is the solution part of the case denoting a list of change operations (i.e., the changes that have been applied to one or more process instances; cf. Def. 2)*
- *rScore indicates how successfully case c has been applied in the past. It is calculated as the sum of the feedback scores.*

All information on process instance changes related to a process schema version S are stored as cases in the associated case-base of S.

Definition 4 (Case–Base). *A case–base* $cb_{T,V}$ *is a tuple (T, V,* $\{c_1, \ldots, c_m\}$, $freq_{T,V}$*)) where*

- *S := S(T,V) denotes the schema version* $cb_{T,V}$ *is currently associated with*
- $\{c_1, \ldots, c_m\}$ *denotes a set of cases*
- $freq_{T,V}(c_i)$ *denotes the frequency* c_i *was reused in connection with schema version S(T,V) in the past, formally:*
  $freq_{T,V}$*:* $\{c_1, \ldots, c_m\} \mapsto \mathbb{N}$

Case Retrieval. When deviations from the predefined process schema become necessary, the user initiates a case retrieval dialog in the CCBR component. The system then assists her in finding already stored similar cases (i.e., change scenarios in our context) by presenting a set of questions. The user may apply a filter to the case-base and/or answer any of the questions in arbitrary order. Filtering is done by specifiying an operation type *opType* and a subject s on which the operation is supposed to operate on. Cases not matching the filter criteria are removed from the displayed list of cases. The system then searches for similar cases by calculating the similarity for each case in the filtered case-base and displays the top n ranked cases (ordered by decreasing similarity) as well as their reputation scores. This is repeated for any other question answered by the user. Case similarity is calculated by dividing the number of correctly answered questions minus the number of incorrectly answered questions by the total number of questions in the case [15].

Case Reuse. When a user decides to reuse an existing case, the actions specified in the solution part of the case are forwarded to and performed by the ADEPT change engine. The reuse counter is incremented and a work item is created for this user for evaluating the ad-hoc change later on to maintain the quality of the case-base.

Adding a New Case. If no similar cases can be found when performing a process instance change, the user adds a new case $c = (pd, \{q_1an_1, \ldots,\}$, sol, 1) to case-base $\mathrm{cb}_{T,V}$. She enters this case by briefly describing the current problem *pd* and by entering a set of question-answer pairs to describe the reasons for the ad-hoc deviation. Question-answer pairs can be entered either by selecting the question from a list of previously defined questions (i.e., reusing questions from existing cases) or, if there is no suitable question in the system, by defining a new one and by giving the appropriate answer. The user then specifies the actions to perform by selecting the desired operation types $opType_1, \ldots, opType_p$. She

further defines the subjects $s_1, \ldots, s_p$ they operate on (e.g., activities and control edges), and provides the parameters for each selected operation. Moreover, the reuse counter of the case is initialized to 1. Finally, the case is stored in case-base $\mathrm{cb}_{T,V}$ of process schema $S = S(T, V)$ and thus immediately made available for future reuse.

Ensuring Semantical Correctness through Evaluation Mechanisms. In our approach we use the concept of reputation to indicate how successfully an ad-hoc modification represented by a case has been applied in the past. Whenever a user adds or reuses a case she is encouraged to provide feedback on the performed process instance change. For this, a work item representing the feedback task is generated and inserted in the worklist of this particular user. She then can later rate the performance of the respective ad-hoc modification either with 2 (highly positive), 1 (positive), 0 (neutral) , -1 (negative) or -2 (highly negative), and may optionally specify additional comments. The reputation score of a case is calculated as the sum of feedback scores regarding the ad-hoc modification specified in this case. While a high reputation score of a case is an indicator for its semantical correctness, negative feedback probably results from problems after performing a process instance change. As ADEPT ensures the syntactical correctness of changes, a negative feedback thus indicates semantical problems. Negative feedback therefore results in an immediate notification of the process engineer, who can then deactivate the case to prevent its further reuse. The case itself, however, remains in the system to allow for learning from failures as well as to maintain traceability.

Example 1. As depicted in Fig. 3 the examination of a patient usually takes place after a preparation step. Before an examination the physician recognizes that the patient suffers from diabetes and he detects several other important risk factors. The physician decides to request an additional lab test for this patient to be performed after activity `Prepare patient` *and before activity* `Examine Patient`*. As the system contains no similar cases, the physician enters a new case describing the situation and the action to be taken (cf. Fig. 2). ADEPT then checks whether inserting activity* `Lab Test` *is possible for the respective process instance, and, if so, applies the specified insert operation to that instance. The latter includes updating the instance markings and all user worklists. If, for example,* `Prepare patient` *is completed and* `Examine Patient` *is activated, this activation will be undone (i.e., respective work items are removed from all user worklists) and the newly inserted activity* `Lab test` *becomes immediately activated. In any case, the newly inserted activity is treated like any other process step, i.e., the same scheduling and monitoring facilities exist.*

When talking with another diabetic patient later on, the physician vaguely remembers that there has been a similar situation before and initiates the CCBR sub-system to retrieve similar cases. For example, as he still remembers that he had performed an additional lab test he selects the `Insert` *operation type as well as the* `Lab Test` *activity to optionally filter the case-base. He then answers the questions presented by the system, finds the previously added case, and reuses it (cf. Fig. 2).*

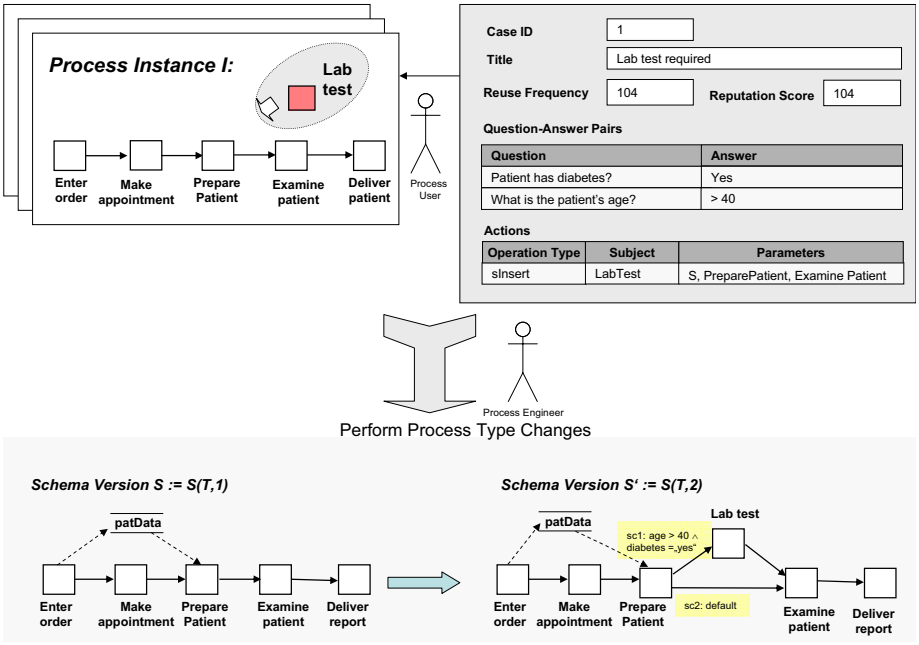

Fig. 3. Perform Process Type Change

4 Process Learning and Seamless Process Evolution

When the same or similar changes occur frequently, the process engineer is notified about the potential need for a process type change (Sect. 4.1). The process engineer can then perform a change of the process type schema and migrate running instances to the new schema version by using the ADEPT framework (Sect. 4.2).

4.1 On Suggesting Process Optimizations

To derive suggestions for process type changes from a collection of previous instance changes we need the following information:

- Case–base $\mathrm{cb}_{T,V} = (\mathrm{T}, \mathrm{V}, \{c_1, ..., c_m\}, \mathrm{freq}_{T,V})$
- $rI_{T,V}$ denoting the number of process instances created from schema version $\mathrm{S} := \mathrm{S}(\mathrm{T}, \mathrm{V})$
- thr denoting a configurable threshold $(0 \leq thr \leq 1)$

If there is a case $c_j \in \mathrm{cb}_{T,V}$ with

$$\frac{freq_{T,V}(c_j)}{rI_{T,V}} \geq thr \tag{1}$$

the process engineer is notified that a process type change should be considered. In this notification the system suggests the solution part sol_j of the respective case c_j as the process type change to be applied, but allows the process engineer to customize it if desired.

Example 2. Let $rI_{T,V} = 10397$, $thr = 0.01$, $cb_{T,V} = \{c_1 =$ *(...,* $sol_1 = \{$`insert(S, X, A, B)`$\}$,*...)*$\}$, *and* $freq_{T,V}(c_1) = 104$. *As* $\frac{freq_{T,V}(c_j)}{rI_{T,V}} = \frac{104}{10397}$ *exceeds threshold* 0.01, *the system suggests to pull change operation sInsert(S, X, A, B) up to the process type level.*

Generally, the situation is more complex, as a certain change operation may have been applied to several instances for different reasons. Note that in our approach this is reflected by sets of different question-answer pairs in separate cases. As a consequence the respective case–base contains distinct cases, i.e., cases with the identical solution parts but different question-answer pairs. Then equation (1) is no longer adequate and must be adapted for a set of cases.

Example 3. Let $rI_{T,V} = 10397$ *and* $thr = 0.01$*;* $cb_{T,V}$ *now becomes:*
$cb_{T,V} = \{c_1 =$ *(...,* $sol_1 = \{$`sInsert(S, X, A, B)`$\}$, *...),*
$c_2 =$ *(...,* $sol_2 = \{$`sInsert(S, X, A, B)`$\}$, *...),*
$c_3 =$ *(...,* $sol_3 = \{$`sInsert(S, X, A, B)`$\}$, *...), ... }* *with*
$freq_{T,V}(c_1) = 48$, $freq_{T,V}(c_2) = 23$, *and* $freq_{T,V}(c_3) = 33$

Exceeding the threshold *thr* can be determined by summing over the frequencies of all cases which have the same solution part, i.e., if there is a set of cases $\mathrm{cb}_{T,V}(\mathrm{sol}) := \{c^{sol} = (..., sol, ...) \in cb_{T,V}) \}$ with

$$\frac{\sum_{c_j \in cb_{T,V}(sol)} freq_{T,V}(c_j)}{rI_{T,V}} \geq thr \tag{2}$$

Thus, in the example above a process type change is indicated and the system suggests *sol* as process type change to the process engineer (e.g., `sInsert(S, X, A, B)`).

Equation (2) still does not reflect the most general scenario as the solution parts of the cases indicating a process type change may not be identical but may have one or more overlapping changes.

Example 4. Assume the following example ($rI_{T,V} = 10397$ *and* $thr = 0.01$*):*
$cb_{T,V} = \{c_1 =$ *(...,* $sol_1 = \{$`sInsert(S, X, A, B), delAct(S, C)`$\}$, *...),*
$c_2 =$ *(...,* $sol_2 = \{$`sInsert(S, X, A, B)`$\}$, *...),*
$c_3 =$ *(...,* $sol_3 = \{$`sInsert(S, X, A, B)`$\}$, *...), ... }* *with*
$freq_{T,V}(c_1) = 48$, $freq_{T,V}(c_2) = 23$, *and* $freq_{T,V}(c_3) = 33$

All cases contain the same change operation `sInsert(S, X, A, B)` but case c_1 contains an additional change operation `delAct(S, C)`. This is not relevant for the evaluation of the case-base and the suggested process type change. (Note that the migration of the respective process instances is handled by the process

schema evolution framework [7].) This can be taken into account by determining the following set

$\mathrm{cb}_{T,V}(\mathrm{sub\_sol}) := \{c = (..., sol, ...) \in cb_{T,V}) \mid \mathrm{sub\_sol} \subseteq \mathrm{sol}\}$

and by applying equation (3):

$$\frac{\sum_{c_j \in cb_{T,V}(sub\_sol)} freq_{T,V}(c_j)}{rI_{T,V}} \geq thr \tag{3}$$

In this case sub_sol is suggested to the process engineer as the process type change to perform (`sInsert(S, X, A, B)` in our example).

The process engineer has to examine the cases that exceeded the threshold as well as the cases with overlapping solution parts in order to decide how to perform the process type change. When a change operation is relevant for all process instances it can be pulled up to the process type level. In most situations the change operations cannot directly be applied to the process schema as the changes have been performed in a particular context. Examining the question-answer pairs allows the process engineer to gain valuable insights into the context of a change operation as each question-answer pair represents a (semantic) condition under which the case was applied. Assume, for example, that a particular change operation has been primarily performed for patients older than 40 years who suffer from diabetes. Therefore, the solution part of the case is not directly pulled up to the process type level, but the process engineer inserts the necessary XOr nodes and transitions (cf. Fig. 3). However, it must be ensured that the necessary data, e.g., patient's age and diabetes (yes/no), is provided to the running process instances after applying the respective process type change, i.e., to guarantee that all necessary data is available within the system.

In the ADEPT approach change operations have formal pre- and postconditions which ensure their correct application, in particular a correct data flow after applying the changes. Therefore, if we can insert a new XOrSplit at the process type level it is guaranteed that all necessary data is available at runtime, as ADEPT allows the application of correct changes only. Data availability can be achieved if either the activities preceding the XOr-Split set the respective data elements (e.g., activity `admit patient` writes patient age and diseases) or parameter provisioning services are inserted directly before the XOrSplit. At runtime these services ask the user for the missing information.

4.2 Process Schema Evolution and Process Instance Migration

Assume that the process engineer decides to apply change operation `cInsert(S, Lab test, Prepare Patient, sc1)` as depicted in Fig. 3 and 4. The challenge is then to migrate the already running process instances to the new schema version S'. As we can see from Fig. 4 we are confronted with different kinds of running process instances: instances still running according to their original schema (unbiased instances, e.g., I_1) and instances which have already been individually modified (biased instances, e.g., I_2 – I_4). We further have to distinguish between biased instances for which their instance–specific change (bias) overlaps the process type change (e.g., I_3, I_4) and biased instances with a disjoint bias

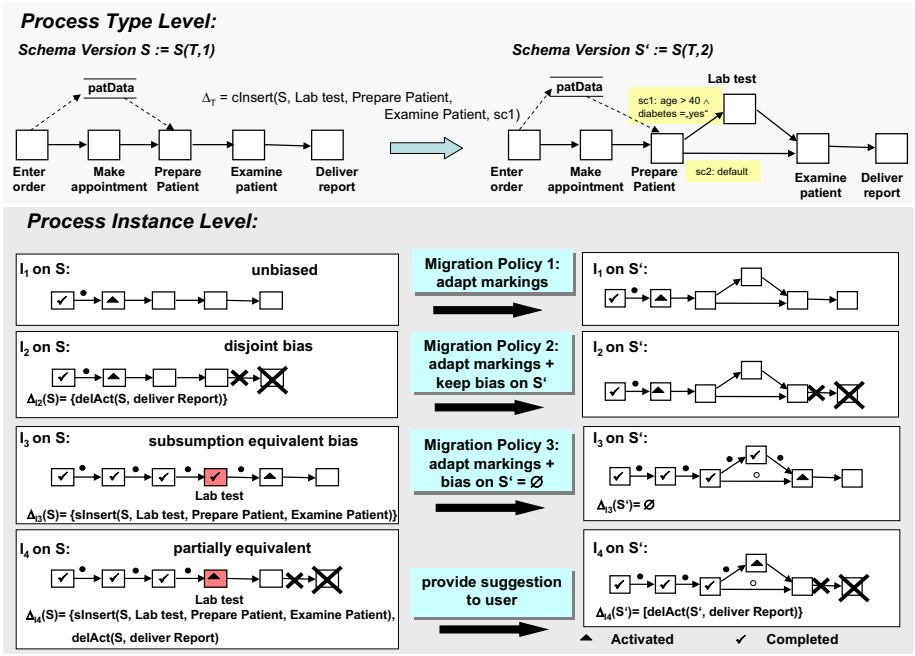

Fig. 4. Process Instance Migration

(e.g., I_2). Process instances with overlapping bias have already anticipated the process type change (cf. Sect. 4.1) and require a different migration policy than the process instances with disjoint bias (for details see [16]).

For unbiased process instances state–related compliance with the new schema version has to be checked [8]. Compliant instances are then migrated to the new schema version by applying marking adaptations (as, for example, depicted in Fig. 4 for instance I_1). Process instances with disjoint bias can be migrated to the new schema version S' if they are compliant regarding their state and their structure [6]. In Fig. 4 instance I_2 has a disjoint bias and is compliant with S'. Therefore I_2 is migrated to the new schema version by adapting its marking and keeping its instance–specific bias in the new schema version (cf. Tab. 2).

The most interesting question is how to deal with the process instances which have totally or partially anticipated the process type change (resulting in an overlap of process type and instance–specific changes). The migration policy to be applied to such instances depends on the particular *degree of overlap* between process type and instance–specific changes, which can be determined precisely by a *hybrid approach* (for details see [7,16]). Table 2 shows the different degrees of overlap and the related migration policies; default migration policies for partially equivalent changes can only be provided in certain situations.

In Fig. 4, for example, instance I_3 would be classified as having a subsumption equivalent bias related to type change Δ_T. Δ_{I_3} and Δ_T both insert activity `lab`

Table 2. Degrees of Overlap Between Changes and Related Migration Policies

Degree of Overlap between Δ_T and Δ_I	*Migration Policy*
Δ_T and Δ_I disjoint, i.e., $\Delta_T \cap \Delta_I = \emptyset$	• apply Δ_T on $S_I := S + \Delta_I$ • migrate I to S' • $\Delta_I(S') = \Delta_I(S)$
Δ_T and Δ_I equivalent, i.e., $\Delta_T \equiv \Delta_I$	• migrate I to S' • $\Delta_I(S') = \emptyset$
Δ_T subsumes Δ_I, i.e., $\Delta_T \prec \Delta_I$	• migrate I to S' • calculate $\Delta_I(S')$
Δ_I subsumes Δ_T, i.e., $\Delta_I \prec \Delta_T$	• migrate I to S' • $\Delta_I(S') = \emptyset$
Δ_T and Δ_I partially equivalent, i.e., $\Delta_T \between \Delta_I$	default policies not always possible → provide suggestion to user

`test`, but Δ_T additionally creates an alternative branching. According to Table 2, I_3 can be migrated to S' by adapting the instance markings of I_3. The instance-specific bias $\Delta_{I_3}(S')$ becomes empty after the migration. Comparing Δ_T with Δ_{I_4} we see that both changes are partially equivalent, thus we cannot provide a default migration strategy [7], but only make a suggestion to the user (cf. Fig. 4). Note that there are optimizations regarding the determination of the precise degree of overlap [7].

In total, we provide a complete framework for migrating process instances to a new schema version even if they have anticipated the type change. This closes the process life cycle depicted in Fig. 1.

5 Case–Base Evolution

Process type changes are accompanied by migrating compliant process instances to the new schema version S' as well as migrating the associated case-base cb to cb'. The challenge is to decide which cases of case-base cb should be transfered to cb' and which ones are already covered by the new schema version S' and can therefore be dropped.

If a case or a group of cases exceeds the predefined threshold the resulting process type change can either be relevant for all process instances or only for a particular subset (cf. Section 4.1). In the former scenario the solution parts of the cases that triggered the change are directly reflected in the new process schema S'. Therefore, cases whose solution part is a subset of Δ_T are not transferred to the new case-base version cb'. Cases whose solution parts are a true superset of Δ_T are presented to the process engineer who then decides whether to transfer these cases or not. Cases without overlapping solution parts are automatically transferred to cb' as they are not covered by S'. In the latter scenario the migration from cb to cb' is more complicated. It involves finding the regions of the process graph that are affected by the process type change Δ_T. In our example the change region corresponds to the subgraph induced by the newly

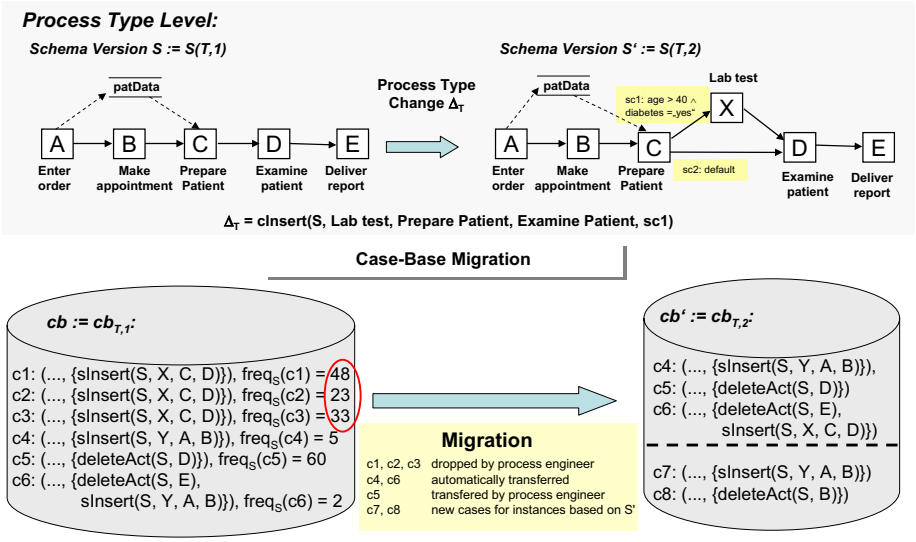

Fig. 5. Case–Base Evolution

inserted activity X and the insertion context (i.e., activities C and D). It can be determined by applying the hybrid approach described in [7]. The cases which contain change operations referring to activities or edges within these regions as subjects or parameters are presented to the process engineer who then can manually transfer relevant cases. All other case are automatically transferred to case-base cb'.

Example 5. As illustrated in Fig. 5 the process engineer has been notified to perform a schema evolution as cases $c_1, ..., c_3$ exceed the predefined threshold value. After migrating schema S to S' the process engineer has to migrate case-base cb to cb' as well. Cases c_4 and c_6 are automatically transferred to cb' as they do not use activities or edges within the affected process graph region. All other cases are presented to the process engineer, who then decides to drop cases $c_1, ..., c_3$ which are already covered by the process type change and to transfer case c_5. Of course, new cases may be added to cb' due to ongoing ad-hoc changes of instances based on S'. Later on, migrating cb' will become necessary when another process schema evolution takes place.

6 Related Work

Process Mining [17] and Delta Analysis [18,19] are techniques to improve the quality of business processes. Though these approaches are very inspiring, they do not answer how they feed the improved process type schemes into the system. To our best knowledge this is accomplished by establishing the mined process schemes as new process type schema versions. Already running process instances

are then completed according to the "old" (suboptimal) process type schema and new instances are started according to the improved one. This leads to a "gap" within the process life cycle which can be closed by applying the derived process optimization to the current process type schema. Already running process instances are then smoothly migrated to the improved process type schema [15,20].

Related work also includes approaches dealing with process schema evolution [1,2,4,21]. However, none of them covers the interplay between process type and process instance changes, i.e., there is no approach which allows to migrate biased process instances to a changed process type schema.

This paper is based on the idea of integrating ADEPT [9] and CCBR [10] (see also [15]). In related work traditional CBR has been applied to configure complex core processes by using process components [22]. Workflows are configured during their instantiation by combining predefined process components in order to reduce the number of possible process variants. As each process instance has to be configured before its start, this approach is more suitable for long-running, complex core processes with a limited number of process instances; the process configuration is similar to a project planning task. Similarily, Madhusudan et al. [23] use CBR to provide workflow modeling support by facilitating the reuse of existing models and their components. In contrast to our approach, CBR techniques are applied to support the modeling of business processes and not their execution.

7 Summary and Outlook

The integration of ADEPT and CBRFlow offers promising perspectives, as process instance changes are enriched with semantic information. This, on the one hand ensures the traceability of instance changes and on the other hand supports users in reusing information about previous instance changes. Furthermore, our approach provides techniques to automatically derive suggestions for process type changes from previously applied instance changes. If the process engineer decides to pull up an instance change to the process type level, already running process instances can be smoothly migrated to the new process schema version. Finally an evolution of the associated case–base is done.

Currently we implement a prototype integrating the concepts of ADEPT and CBRFlow and plan to evaluate the resulting prototype in different application scenarios. Future work will focus on the semantic compliance of process type and process instance changes when they are concurrently applied to the same process schema. In this context the representation and the evaluation of semantic information stored for process changes are challenging research topics.

References

1. v.d. Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoret. Comp. Science **270** (2002) 125–203
2. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: Proc. Int'l COOCS'95, Milpitas, CA (1995) 10–21

3. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. DKE **50** (2004) 9–34
4. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: Proc. HICSS-34. (2001)
5. Reichert, M., Rinderle, S., Dadam, P.: On the common support of workflow type and instance changes under correctness constraints. In: Proc. Int'l CoopIS'03. LNCS 2888, Catania, Italy (2003) 407–425
6. Rinderle, S., Reichert, M., Dadam, P.: On dealing with structural conflicts between process type and instance changes. In: Proc. BPM'04, Potsdam (2004) 274–289
7. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, University of Ulm (2004)
8. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. DPD **16** (2004) 91–116
9. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. JIIS **10** (1998) 93–129
10. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In: Proc. ECCBR'04, Madrid (2004) 434–448
11. Leymann, F., Altenhuber, W.: Managing business processes as an information ressource. IBM Systems Journal **33** (1994) 326–348
12. Kolodner, J.L.: Case-Based Reasoning. Morgan Kaufmann (1993)
13. A. Aamodt, E.P.: Case-based reasoning: Foundational issues, methodological variations and system approaches. AI Communications **7** (1994) 39–59
14. Aha, D.W., Muñoz-Avila, H.: Introduction: Interactive case-based reasoning. Applied Intelligence **14** (2001) 7–8
15. Weber, B., Rinderle, S., Wild, W., Reichert, M.: CCBR–driven business process evolution. In: Proc. Int. Conf. on Cased based Reasoning (ICCBR'05), Chicago (2005)
16. Rinderle, S., Reichert, M., Dadam, P.: Disjoint and overlapping process changes: Challenges, solutions, applications. In: Proc. CoopIS'04, Cyprus (2004) 101–120
17. v.d. Aalst, W., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow mining: A survey of issues and approaches. DKE **27** (2003) 237–267
18. v.d. Aalst, W.: Inheritance of business processes: A journey visiting four notorious problems. In: Proc. Petri Net Technology for Communication Based Systems. LNCS 2472 (2003) 383–408
19. Guth, V., Oberweis, A.: Delta analysis of petri net based models for business processes. In: Proc. Applied Informatics. (1997) 23–32
20. Weber, B., Reichert, M., Rinderle, S., Wild, W.: Towards a framework for the agile mining of business processes. In: Proc. of BPM 05 BPI workshop. (2005)
21. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. DKE **24** (1998) 211–238
22. Wargitsch, C., Wewers, T., Theisinger, F.: An organizational memory-based approach for an evolutionary workflow management system. In: Proc. HICCS-31. (1998) 174–183
23. Madhusudan, T., Zhao, J.: A case-based framework for workflow model management. In: Proc. Int'l Conf. BPM'03, Eindhoven (2003) 354–369